UNIVERSITY CENTER
77 — BN
NO REFUND WITHOUT
LABEL ATTACHED
8C 1.50

User's Guide to

P L / C

The Cornell Compiler for PL/I

Release 7

DEPARTMENT OF COMPUTER SCIENCE

CORNELL UNIVERSITY

ITHACA, NEW YORK 14850

User's Guide to PL/C

The Cornell Compiler for PL/I

Release 7
OS and DOS Versions

June 1974

Department of Computer Science
Cornell University
Ithaca, N. Y. 14850

This material is not copyrighted. It may be reproduced, in whole or in part, for any purpose.

PL/C Project:
S. Worona, Manager
M. Bodenstein
R. Conway
J. Bayard
D. Gries
P. Harter
P. Hellberg
Chalmers Univ.
G. Hillenbrand
T. London
W. Maxwell
C. Moore
P. Krug
Siemens Akt.
T. Wilcox
Univ. of Illinois

Previous Contributors:
H. Morgan
R. Wagner
H. Cabassa
P. Dormont
H. Elder
R. Fisher
R. Holt
T. Kahne
S. Lisberger
T. Vari
N. Weiderman
W. Worley
M. Zelkowitz

PL/C: The Cornell Compiler for PL/I

PL/C was designed to permit efficient instruction in the PL/I language. It provides high-speed compilation, reasonably efficient execution, extraordinary diagnostic assistance and upward compatibility with the IBM F-level compiler.

PL/C does not support the full PL/I language. The major omissions in Release 7 are:

- list processing
- multi-tasking
- compile-time facilities (except for INCLUDE and a non-PL/I-type macro processor)
- direct access auxiliary files

There are other minor omissions described in Sections D - H of this Guide.

Some features have been added to PL/C that are not part of PL/I. Except for the MACRO feature (new in Release 7) described in Section I, and a built-in function to generate random numbers these features are intended to provide additional diagnostic facilities. They include special options on the PUT statement, the FLOW condition, statements to control FLOW and CHECK printing, and pseudo-comments that can optionally be converted to source text. These pseudo-comments can be used to shelter the incompatible PL/C features so that a program can still be run under the PL/I-F compiler.

This User's Guide is not intended to teach a beginner to write programs in PL/C. It assumes that the reader already knows PL/I, or is in the process of learning the language from another source. This Guide describes how PL/C differs from PL/I, and gives information necessary to use PL/C and interpret output provided by PL/C. Except as noted in this Guide PL/C is consistent and compatible with PL/I as defined by the IBM F-level compiler. (See IBM references in Section L.) This Guide covers both the OS and DOS versions of PL/C. Except as noted to the contrary, material applies to both versions. The contents of the Guide are organized as follows:
(* denotes major change from Release 6.5)

A. Deck structure and control cards .. 3
* B. Options .. 3
* C. Error messages .. 5
D. Summary of differences between PL/C and PL/I .. 14
E. Statements .. 15
F. Attributes .. 18
G. Built-in functions and pseudo-variables .. 19
H. Conditions .. 20
* I. The PL/C MACRO feature .. 22
J. The INCLUDE feature (not in DOS PL/C) .. 23
* K. Efficient programming in PL/C .. 24
* L. References and textbooks .. 24

Section A. Deck Structure and Control Cards

There are three principal control cards:
*PL/C - to begin a program and specify options
*PROCESS - to separate external procedures and specify options
*DATA - to denote the beginning of input data
(not required if no data is present)

Other control cards give additional flexibility in option specification (see Section B), control the INCLUDE feature (see Section J), and control the MACRO feature (see Section I).

The control cards described here are only those cards required by PL/C. The operating system may require other control cards to precede those shown to invoke the PL/C compiler, and after those shown to close the job. PL/C will use whatever region is passed to it by the operating system. With the usual (overlayed) version of PL/C a minimum region of approximately 100K is required. 110K is more reasonable and will accommodate a program of about 200 statements. As the program length and number of identifiers increase more space is required.

PL/C is designed to process a group of independent programs that will all look like a single job to the operating system. Each *PL/C card reinitializes the compiler to begin a new program.

PL/C control cards are recognized by a * in column 1. This must always be in column 1 -- it is not affected by the SORMGIN option. This "recognition character" is an installation option and may be any non-blank character, but this Guide will describe all control cards as if the character is a *.

A program consisting of a single external procedure and requiring no data would simply be the following:

```
*PL/C      options
    source program cards
```

A program with two external procedures and data would be:

```
*PL/C      options
    source cards for external procedure
*PROCESS      options
    source cards for external procedure
*DATA
    data cards
```

Note that the control cards are distinct from both program and data cards. Program statements must not begin on a control card; data cannot begin on the *DATA card.

Section B. Options

User options may be specified on three different types of control cards:
*PL/C - one such card, defines beginning of new program
*PROCESS - required between external procedures
*OPTIONS - permitted anywhere after *PL/C and before *DATA (or end-of-file)

The options on these cards are separated by blanks and/or commas, and may be continued onto subsequent cards. The continuation of any of these cards has a * in column 1 with columns 2 and 3 blank. An individual option may not be split over a card boundary. Options may be abbreviated (or misspelled) as only a few key letters are actually significant, as indicated after the option in the listing below. The prefix letters N or NO designate negated options. Certain of the options can only be given on the *PL/C card, as noted.

Options specified on the *PL/C card, and the default values for the options not specified, are in effect throughout the program, except as overridden by specifications on subsequent *PROCESS and *OPTIONS cards. After each external procedure options are reset to the "global" *PL/C and default values. If the *PROCESS card changes any options, the changes affect only the one external procedure following that card. Similarly, specifying an option on a *OPTIONS card changes the option only for that portion of the one external procedure following the *OPTIONS card.

In the listing below the supplied default value of the option is underlined, but these choices are easily changed by each installation, and may therefore differ from what is shown here. In addition, installations have the capability of over-riding options so that user specification of those options may be ineffective.

ATR, NOATR, A
Produce attribute listing.

ALIST, NOALIST, AL
Produce assembler listing of generated object code.

AUXIO=n, AU (on *PL/C only)
Limit on number of auxiliary input/output operations.
Supplied default n=10000.

BNDRY, NOBNDRY, B
Strings and comments limited by card boundary.

CMNTS, CMNTS=(n1,n2,...), NOCMNTS, C
Contents of comments beginning with : considered source text. If parameter(s) are given (1<=ni<=7) comments beginning with ni are also considered source text.

CMPRS, NOCPRS, CP
Source listing to be given in compressed form (certain page ejects replaced by 3 line skips).

DUMP, DUMP=(d1,d2,...), NODUMP, D (on *PL/C only)
Produce post-mortem dump.
Dump options are:
- BLOCKS, B
  Traceback of blocks active at termination.
- SCALARS, S
  Final values of scalar variables in active blocks. (Implies B.)
- ARRAYS, A
  Final values of arrays in active blocks. (Implies S and B.)
- FLOW, F
  History of last 16 transfers of control.
- LABELS, L
  List of labels with frequency of encounter.
- ENTRIES, E
  List of entry-names with frequency of call.
- REPORT, R
  Statistics on run (time, core usage, auxiliary I/O operations, etc.)
- UNREAD, U
  List of first 5 or fewer unread data cards.
- Depth
  An integer giving limit on number of active blocks for B, S and A dump options. If 0 is given, depth is unlimited.

Supplied default DUMP options are (B,S,F,L,E,R,U,0).

DUMPE, DUMPE=(d1,d2,...), NODUMPE, DE (on *PL/C only)
Produce post-mortem dump only if error was encountered during execution.
Supplied default DUMPE options are (B,S,F,L,E,R,U,0).

DUMPT, DUMPT=(d1,d2,...), NODUMPT, DT (on *PL/C only)
Produce post-mortem dump only if execution was terminated by an error.
Supplied default DUMPT options are (B,S,F,L,E,R,U,0).

ERRORS=(c,r), E (on *PL/C only)
Suppress execution if c or more compile errors.
If c=0 suppress execution unconditionally.
Terminate execution after r runtime errors.
If r=0 there is no limit on runtime errors.
Supplied default c=50, r=50.

FLAGE, FLAGW, FE, FW
FLAGW prints both warnings and error messages.
FLAGE suppresses warnings.

FREE, NOFREE, FR (on *PL/C only)
Permit NOSUBSCRIPTRANGE prefix to be effective.

HDRPG, NOHDRPG, H (on *PL/C only)
Print header/separator page before program.

ID='name', I (on *PL/C only)
Program identification name (20 characters maximum).
Supplied default name = '*** NO ID ***'

LINES=n, L (on *PL/C only)
Maximum number of lines to be printed.
Supplied default n=2000.

LINECT=n, LC
Lines to be printed per page during compilation.
Supplied default n=60.

MCALL, NOMCALL, MC
Print macro calls.

MTEXT, NOMTEXT, MT
Print macro text expansion.

M91, NOM91, M9 (on *PL/C only)
Generate code to run on 360 Model 91.

OPLIST, NOOPLIST, O
Print list of options in effect.

PAGES=n, P (on *PL/C only)
Maximum number of pages to be printed.
Supplied default n=30.

SORMGIN=(s,e), SORMGIN=(s,e,c), SM
Establish source card margins:
- s is first column scanned; supplied default s=2.
- e is last column scanned; supplied default e=72.
- c is carriage control column; supplied default c=1.

SOURCE, NOSOURCE, S
Print source program listing.

TIME=(m,s), T (on *PL/C only)
Time limit (compilation + execution).
- m is minutes; blank is equivalent to 0; supplied default m=0.
- s is seconds; blank is equivalent to 0; supplied default s=10.

TABSIZE=n, TS (on *PL/C only)
Determines amount of PL/C region allocated to symbol table. n given in fullwords.
Supplied default is 1/2 of usable area.

UDEF, NOUDEF, U (on *PL/C only)
Monitor use of uninitialized variables.

XREF, NOXREF, X
Produce cross-reference listing.

Section C. Error Messages

In the text of the following messages, the upper-case letters are shown as they appear on the PL/C program listing, but the lower-case words are replaced by variable information as follows:

- iden -a variable or label name will be printed
- string -a character string will be printed
- number -a fixed or floating point number will be printed
- rtn -the name of a subroutine will be printed
- line -a statement number will be printed
- attribute -an attribute will be printed

The error message numbers in PL/C are prefixed with a two letter code indicating the phase in which the error occurred. The codes and their meanings are:

- SY or MD syntactic analysis phase
  (MD for errors during macro definitions)
- SM semantic analysis phase
- XR cross reference phase
- CG code generation phase
- EX execution phase

One group of messages, numbered E2-EA, can appear in any phase of the compiler, hence will be given with a prefix for the phase in which the error occurs. All of the other messages appear in only one phase and are grouped in the list below by phase and prefix.

A line beginning PROGRAM CHECK or COMPILER ERROR indicates a problem in the PL/C compiler and not a user error (although it most often occurs in response to some user error.) Please bring such programs to the attention of your systems programming staff so that we will be notified and the problem can be remedied.

Variable Prefix Errors that can occur in any Phase

NUMBER	MESSAGE
E2	ERROR LIMIT EXCEEDED
E3	LINE LIMIT EXCEEDED
E4	PAGE LIMIT EXCEEDED
E5	TIME LIMIT EXCEEDED
E6	TIME LIMIT EXCEEDED - PROBABLE COMPILER LOOP (the specified time limit has expired and an additional second has passed without the completion of a source statement. This is probably a compiler error, see beginning of Section C.)
E7	I/O ERROR: string (the operating system reported a "SYNAD"-type I/O error. The string given is the SYNADAF message. See appropriate IBM system manuals.)
E8	UNABLE TO PROCESS INCLUDE COMMAND (see Section J)
E9	SYMBOL-TABLE OVERFLOW. USE LARGER REGION OR INCREASE TABSIZE (see TABSIZE option in Section B)
EA	STRING TOO LONG FOR LINE IN ABOVE MESSAGE (OR COMPILER ERROR) (PL/C tried to print an error message containing a string longer than an output line. The message is terminated and execution continues.)

SY or MD Prefix Errors during the Syntactic Analysis Phase

In some cases, particularly in declarations, errors are discovered too late in the analysis of the statement for PL/C to conveniently correct the text of the statement. The internal form of the program has been altered to a correct construction (often just a null statement) but the usual display of corrected source text is omitted. Messages that frequently are issued in such circumstances are marked with a * in the following list.

NUMBER	MESSAGE
00	MISSPELLED KEYWORD (apparent misspelling of one of the reserved keywords. See list under DECLARE in Section E.)
01	EXTRA ((deleted)
02	MISSING ((supplied)
03	EXTRA) (deleted)
04	MISSING) (supplied)
05	EXTRA COMMA (deleted)
06	MISSING COMMA (supplied)
07	EXTRA SEMI-COLON (deleted)
08	MISSING SEMI-COLON (OR MISUSE OF RESERVED WORD) (A semi-colon is supplied. A frequent cause of this error is the use of a reserved keyword as an identifier. This forces the start of a new statement so the statement prematurely ended apparently lacks a semi-colon. The user may need to choose a non-reserved identifier rather than supply a semi-colon)
09	MISSING : (supplied)

```
0A      MISSING =
           (supplied)
0B *    IMPROPER ATTRIBUTE ON PARAMETER
0C      INEFFECTIVE IF
           (warning; a pointless IF statement has been given.
           The THEN unit is null and there is no ELSE unit.)
0D *    IMPROPER ENTRY/RETURNS ATTRIBUTE
0E      MISSING END
           (*PROCESS, *DATA, or end of program encountered
           with a block still open; END supplied)
0F      MISSING KEYWORD
           (the required keyword is supplied)
10      INCOMPLETE EXPRESSION
11      MISSING EXPRESSION
12      MISSING VARIABLE
13      MISSING ARGUMENT, 1 SUPPLIED
14      EMPTY LIST
15      IMPROPER NOT
           (¬ cannot be used as a binary operator,
            ¬= substituted)
16      IMPROPER ELEMENT
           (an element that won't fit has been discarded)
17      IMPROPER SYNTAX, TRANSLATION SUSPENDED
           (The statement has been abandoned and replaced by
           a null statement. PL/C scans ahead for a semi-colon
           or reserved word to start the next statement.)
18      INCONSISTENT OPTION, STATEMENT DELETED
           (After the statement was completed, it was found
           to contain inconsistent options. It is deleted and
           replaced by a null statement.)
19      NOT ENOUGH CORE, TRY LARGER REGION
           (program will not be executed as is)
1A      NESTING TOO DEEP
           (nesting depth exceeds capacity of PL/C. See
           restrictions in Section D.)
1B      INACCESSIBLE STATEMENT
           (warning; this statement cannot be reached in
           execution)
1C      MISSING MAIN PROC
           (warning; no procedure has OPTIONS(MAIN)
           phrase; the first procedure is assumed to be MAIN)
1D      MISSING PROCEDURE STATEMENT
           (A statement is not contained in an external proc,
           or an identifier declared ENTRY appears as a
           non-entry label. A PROC statement is supplied
           by PL/C. This condition is sometimes a by-product
           of another error; for example, an extra END in the
           interior of a procedure ends it prematurely and
           causes subsequent statements to be apparently
           outside of any procedure.)
1E      MISSING *PROCESS (OR EXTRA END)
           (either the required *PROCESS between external
           procedures has been omitted, or an extra END has
           prematurely ended a procedure. PL/C supplies a
           *PROCESS)
1F      MISPLACED ENTRY STATEMENT
           (ENTRY cannot be in a BEGIN block or in an
           iterative DO loop)
20      IMPROPER OPTION(S)
           (improper option on statement, or invalid option
           on *PL/C, *PROCESS or *OPTIONS card. See Section B.)
21      IMPROPER FORMAT ITEM
22      IMPROPER I/O PHRASE
23      IMPROPER TO PHRASE
24      IMPROPER BY PHRASE
25      IMPROPER WHILE PHRASE
26      IMPROPER SPECIFICATION
           (error in iteration specification of DO statement)
27      MULTIPLE DECLARATION
           (This identifier has already been used in a way that
           precludes its appearance here. In most contexts the
           identifier is replaced by a new identifier generated
           by PL/C.)
28 *    IMPROPER attribute ATTRIBUTE FOR iden
           (the attribute indicated cannot be applied to this
           identifier, usually because of previous attributes)
29      IMPROPER FACTORING
2A *    IMPROPER DIMENSION
2B *    IMPROPER PRECISION
2C *    IMPROPER SCALE
2D *    IMPROPER VARYING ATTRIBUTE
           (string type not specified; VARYING deleted)
2E      IMPROPER FILE-NAME
           (the identifier in a FILE phrase is not a valid
           file-name)
2F      EXTERNAL NAME TOO LONG
           (warning, PL/I allows a maximum of 7 characters
           for an identifier in this context)
30 *    IMPROPER INIT ATTRIBUTE
           (INITIAL is incompatible with previous attributes)
31 *    IMPROPER STRUCTURE LEVEL
           (This may be improper construction of a structure
           declaration.  It also arises from any stray
           integer in a faulty declaration.)
32 *    IMPROPER ATTRIBUTE IN STRUCTURE
           (major and minor structure names cannot have type
           attributes, leaves cannot have storage class attr)
33      TOO MANY IDENTIFIERS
           (there is a PL/C limit of 88 identifiers in a
           single factor or structure; program is not executed)
34      IMPROPER THEN OR ELSE
           (THEN or ELSE given on a statement not following
           an IF)
35      IMPROPER THEN OR ELSE UNIT
           (This statement is not allowed as a THEN|ELSE
           unit. ELSE is deleted; null statement inserted
           after THEN.)
36      MISSING THEN
           (THEN supplied)
```

37 IMPROPER CHECK OR NOCHECK
(prefix applied to a statement other than BEGIN or PROCEDURE; prefix deleted)

38 IMPROPER PREFIX ORDER
(warning- executes correctly under PL/C but incompatible with PL/I-F)

39 EXTRA LABEL

3A IMPROPER LABEL

3B MISSING LABEL OR ENTRY NAME
(a new identifier is generated by PL/C)

3C IMPROPER ON-CONDITION
(often triggered by an extra left paren, as for example around the left side of an assignment, which is taken to be the beginning of a condition prefix)

3D IMPROPER ON-UNIT
(This statement is not allowed as a simple ON-unit. BEGIN and END are supplied.)

3E IMPROPER SPACE
('NO' is improperly separated from rest of keyword. Space is removed. Example: NO CHECK becomes NOCHECK)

3F PL/I FEATURE NOT IN PL/C
(the feature used is not included in current PL/C)

40 FEATURES INCOMPATIBLE WITH PL/I-F HAVE BEEN USED
(warning; an incompatible PL/C feature has been used and is not enclosed in a pseudo-comment. The program will not be accepted by PL/I-F.)

41 INCOMPATIBLE OPTION

42 MISSING OPTION
(a "required" option is not given. For example, ENV requires CONSECUTIVE)

43 IMPROPER LOGICAL UNIT
(DOS only: improper logical unit in MEDIUM option of ENVIRONMENT attribute)

44 MISSING DECIMAL INTEGER

45 NON-* BOUND/LENGTH FIELD
(* and not an expression must be given for subscript bound or string length in this context)

46 DECLARATION FOR ENTRY iden DOES NOT AGREE WITH CORRESPONDING PROCEDURE OR ENTRY POINT

47 PROCEDURE iden IS NOT PRESENT
(iden has been declared as an entry-name, but no corresponding procedure definition appears)

48 *-LENGTH NOT ALLOWED. 256 USED

49 TOO MANY DIGITS IN EXPONENT

4A ILLEGAL EXPONENT
(illegal character appeared after E. A space is inserted before the E.)

4B ILLEGAL BINARY NUMBER
(it is treated as decimal)

4C ILLEGAL USE OF COLUMN 1 ON CARD
(if SM=(2,X,1) the value in column 1 is concatenated with the string beginning in column 2)

4D */ NOT IN COMMENT
(it is ignored)

4E NAME > 31 CHARACTERS
(the first 256 or fewer characters are used)

4F ILLEGAL CHARACTER
(it is ignored)

50 STRING CONSTANT RUNS ACROSS CARD BOUNDARY
(a ' is supplied)

51 IMBEDDED BLANK(S) IN OPERATOR
(the blanks are ignored. Example: * * becomes **)

52 COMMENT RUNS OVER CARD BOUNDARY
(the comment is terminated at the end of the card)

53 2 DECIMAL POINTS IN NUMBER
(number is ended before the 2nd decimal point)

54 EXPONENT RUNS ACROSS CARD BOUNDARY
(it is ignored)

55 SPACE MISSING BETWEEN NUMBER AND LETTER
(required space is supplied)

56 MISSING */ BEFORE END OF FILE OR CONTROL CARD

57 INVALID BIT STRING

58 MISSING QUOTE BEFORE END OF FILE OR CONTROL CARD

59 STRING LENGTH > 255

5A MISPLACED *MEND CARD

5B ERROR STACK OVERFLOW - MESSAGE(S) LOST

5C TABSIZE TOO LARGE. DEFAULT USED

5D OPTION(S) NOT ALLOWED AT THIS INSTALLATION
(option specified on *PL/C, *PROCESS or *OPTIONS card is prohibited by this installation)

5E TOO MANY SIGNIFICANT DIGITS, 16 USED

5F TOO MANY SIGNIFICANT DIGITS, 53 USED

60 EXPONENT TOO LARGE

61 NUMBER TOO SMALL

62 NUMBER TOO LARGE/SMALL, number USED
(0 or 10**75 is supplied, as appropriate)

63 MISSING *MEND BEFORE END OF FILE OR CONTROL CARD

64 MISSING MACRO NAME

65 MISSING %; BEFORE END OF FILE OR CONTROL CARD

66 MACRO NAME ILLEGAL OR ALREADY IN USE

67 MISSING PARAMETER NAME IN MACRO DEFINITION

68 MACRO PARAMETER NAME > 31 CHARACTERS. FIRST 31 USED

69 TOO MANY MACRO PARAMETERS. LIST TRUNCATED

6A MACRO PARAMETER NAME APPEARS TWICE IN LIST

6B SYMBOL TABLE AREA OVERFLOW. INCREASE CORE AVAILABLE

6C ILLEGAL CHARACTER(S) ON CARD. BLANK(S) USED

6D MACRO EXPANSION CAUSES REPRINTING OF ABOVE LINE

6E DYNAMIC CORE OVERFLOW DURING MACRO EXPANSION.
INCREASE REGION

6F SCANNING OF ABOVE LINE TERMINATES IN COLUMN number

70 COMPILER ERROR--ILLEGAL INTERNAL MACRO PARM ID.
(compiler error, see beginning of Section C)

71 MISSING (IN MACRO CALL

72 MACRO ARGUMENT > 256 CHARACTERS. FIRST 256 USED

73 TOO FEW ARGUMENTS IN MACRO CALL.
NULL STRING(S) SUPPLIED

74 MISSING COMMA IN MACRO CALL

75 MISSING) IN MACRO CALL

76 END OF FILE OR CONTROL CARD WITHIN MACRO CALL

SM Prefix Errors during the Semantic Analysis Phase

When statements in error are reconstructed during the semantic analysis phase, an additional line is printed, labeled DECLARED IN BLOCK. This line specifies the block in which each variable has been declared.

NUMBER	MESSAGE
40	VARIABLE NOT PERMITTED (must have constant in this context)
41	WRONG TYPE FOR EXPRESSION (expression types are arithmetic, string, label, or file, and the wrong one has been used here.)
42	WRONG STRUCTURE OR DIMENSIONALITY FOR EXPRESSION (scalar needed where an array or matching structure has been used)
43	ILLEGAL SUBSCRIPTING (subscripts are not allowed in certain contexts, e.g., GET DATA)
44	ILLEGAL USE OF PSEUDO-VARIABLES (e.g., CHECK prefixes, GET/PUT DATA)
45	NAME NEEDED (in this context, e.g., initializing label constants
46	ENTRY-NAME NEEDED (CALL must have entry name)
47	NO STRUCTURE APPEARED (in BY NAME assignment)
48	STRUCTURES DO NOT MATCH (in BY NAME assignment)
49	FUNCTION ARGUMENTS MISSING
4A	OPERAND OF BINARY OPERATOR string HAS IMPROPER TYPE
4B	OPERANDS OF BINARY OPERATOR string DISAGREE IN TYPE, STRUCTURE OR DIMENSIONALITY
4C	OPERAND OF UNARY OPERATOR string HAS IMPROPER TYPE
4D	SUBSCRIPT number OF iden NOT NUMERIC
4E	iden HAS TOO MANY SUBSCRIPTS. SUBSCRIPT LIST DELETED
4F	iden HAS TOO FEW SUBSCRIPTS. SUBSCRIPT LIST DELETED
50	NAME NEVER DECLARED, OR AMBIGUOUSLY QUALIFIED (expression replaced or CALL deleted)
51	SUBSCRIPT number OF iden NOT SCALAR
52	iden HAS TOO MANY ARGUMENTS. FUNCTION REFERENCE DELETED
53	ARGUMENT number OF FUNCTION iden DISAGREES WITH CORRESPONDING PARAMETER
54	iden HAS TOO FEW ARGUMENTS. FUNCTION REFERENCE DELETED
55	ARGUMENT number OF FUNCTION iden WAS *. ILLEGAL ARGUMENT.
56	TABLE OVERFLOW. STATEMENT DELETED (processing expression. Try larger region.)
57	TABLE OVERFLOW. STATEMENT DELETED (processing expression skeleton, try larger region.)
58	TABLE OVERFLOW. STATEMENT DELETED (processing expression tree. Try larger region.)
59	TABLE OVERFLOW. STATEMENT DELETED (processing entry parameter. Try larger region.)
5A	iden HAS WRONG # OF SUBSCRIPTS (BY NAME assignment. Structures don't match)
5B	MISMATCHED DIMENSIONALITY (BY NAME assignment. Structures don't match)
5C	ILLEGAL LABEL VARIABLE iden (subscripted label not declared in block)
5D	ILLEGAL ASSIGNMENT TARGET
5E	ASSIGNMENT SOURCE INCOMPATIBLE WITH TARGET
5F	MAJOR STRUCTURE NAME NEEDED
60	DEFAULT ATTRIBUTES FOR ENTRY NAME iden CONFLICT WITH RETURNS OPTION IN STMT line
61	iden IS ASSUMED A VARIABLE, NOT A BUILT-IN FUNCTION

XR Prefix Errors during the Cross-Reference Phase

NUMBER	MESSAGE
62	NOT ENOUGH CORE FOR CROSS-REFERENCE (use larger region)
63	CROSS REFERENCE ABBREVIATED DUE TO LACK OF SPACE (use larger region)
64	COMPILER ERROR IN XREF PHASE--INVALID STATEMENT CODE (a compiler error, see beginning of Section C)

CG Prefix Errors during the Code Generation Phase

NUMBER	MESSAGE
00	FORMAT WILL BE EXECUTED ONLY ONCE (the format specification of the EDIT statement does not contain any format items which would cause data to be transferred between the I/O list and the I/O buffer --i.e. No A,B,C,E,F or R format item. If run under PL/I-F the program would loop)
01	CONSTANT BOUND, LENGTH, SUBSCRIPT OR ITERATION FACTOR EXCEEDS 32767 IN MAGNITUDE. 10 IS USED. (PL/I language restriction)
02	WORKSPACE OVERFLOW IN STATEMENT PROCESSING (the combined nesting of BEGIN and PROCEDURE blocks, iterative DO groups, and IF statements is too deep for the code generation phase. The rest of the program is not scanned for code generation errors. Increasing the region size will not help. The nesting depth must be reduced)
03	iden REQUIRES TOO MUCH SPACE. UPPER BOUND OF SUBSCRIPT number IS SET TO LOWER BOUND. (more than 2**31 bytes would be required for the array as declared)
04	PRIMARY DATA STORAGE AREA FOR BLOCK # number EXCEEDS SIZE LIMIT BY number BYTES. (primary data storage does not include space for arrays or strings. Try adding some more BEGIN blocks)
05	LENGTH OF iden (number) IS NOT IN PROPER

RANGE. 80 IS USED.
(length is <0 or > 256)

06 iden REQUIRES TOO MUCH SPACE. LOWER BOUND OF SUBSCRIPT number IS SET TO ZERO.
(the array element with all subscripts zero must be within 2**31 bytes of the array element with all subscripts at their lower bound. Move the lower bounds closer to zero and resubmit)

07 VALUE IN RETURN HAS ATTRIBUTES INCOMPATIBLE WITH ONE OR MORE ENTRY POINTS.
(warning. Execution errors may be generated if this return is used)

08 SEVERE ERRORS. EXECUTION DELETED.
(a previous code generation error has made it impossible to continue into execution. All code generation errors have been reported)

09 CONVERSION REQUIRED TO MATCH ARGUMENT iden OF iden
(warning. PL/C has generated code to convert the argument of a procedure call so that the attributes of the value passed will match the attributes of the corresponding parameter. PL/I-F would not do this conversion because the attributes of the parameter have not been specified in an ENTRY declaration)

0A SCALAR ARGUMENT SUPPLIED TO AGGREGATE PARAMETER iden OF iden . (1:10 USED FOR ALL BOUNDS.)
(this is a PL/C restriction, see Section E. Assign the constant to an array with the proper bounds and pass that array to the procedure.)

0B WORKSPACE OVERFLOW IN EXPRESSION PROCESSING
(either the situation which would generate error CG02 exists or the nesting of array expressions, array subscripting, function references, or parenthesized expressions is too deep. Simplify the expression. Increasing region size will not help)

0C NO FILE SPECIFIED. SYSIN/SYSPRINT ASSUMED.
(warning)

0D iden IS A PARAMETER IN I/O LIST
(warning. PL/I-F does not allow parameters in DATA- directed I/O lists nor in CHECK prefixes. PL/C will accept the parameter)

0E BOTH FORMS OF INITIALIZATION USED FOR LABEL VARIABLE iden
(PL/I-F does not permit a LABEL variable to be initialized via both the INITIAL attribute and subscripted statement label constants. Both forms are accepted by PL/C. Where there is conflict the INITIAL attribute takes precedence)

0F STORAGE CAPACITY IS EXCEEDED
(object code exceeds available space, specify larger region)

10 ILLEGAL COMPLEX COMPARE. REAL PARTS WILL BE COMPARED

11 iden IS ILLEGAL OPERAND IN INITIAL, LENGTH OR DIMENSION ATTRIBUTE OF STATIC VARIABLE. constant IS USED.
(the bounds, lengths, and iteration factors used with a STATIC or EXTERNAL variable must be optionally signed decimal constants. A non-constant has appeared in this context and has been replaced by a constant of appropriate type)

12 NON-CONSTANT OPERAND(iden) IN INITIAL, LENGTH OR DIMENSION ATTRIBUTE OF STATIC VARIABLE.
(warning: a STATIC or EXTERNAL variable, BUILT-IN function, EXTERNAL user-defined function has been used in the bounds, length, or iteration factor for a STATIC/EXTERNAL variable. This is not allowed in PL/I. PL/C uses the value of the operand in error)

13 PL/C BUILT-IN FUNCTION USED.
(warning. A builtin function that is not included in PL/I-F has been used)

14 ARGUMENT TO MAX OR MIN IS COMPLEX. REAL PART IS USED.

15 NO SCALE FACTOR ARGUMENT APPEARED. RESULT IS SET FLOAT.
(see explanation of error CG16)

16 UNNECESSARY SCALE FACTOR ARGUMENT APPEARED. RESULT IS SET FIXED.
(for ADD, DIVIDE or MULTIPLY, both a precision argument (P) and a scale factor argument (Q) must be present if the result is to have FIXED scale. Only argument P may appear if the result is to have FLOAT scale. If either requirement is violated, PL/C converts the argument to the scale implied by the number of arguments given)

17 ARGUMENT SHOULD BE A CONSTANT. 10 IS USED.
(certain arguments to the builtin functions ADD, BINARY, DECIMAL, DIVIDE, FIXED, FLOAT, MULTIPLY, PRECISION and ROUND must be decimal constants in PL/I-F)

18 ABS(ARGUMENT) > 32767. 10 IS USED.
(constant arguments to builtin functions mentioned in explanation of error CG17 must be less than 32768 in magnitude)

19 ARGUMENT SHOULD BE REAL. IMAGINARY PART IS USED.
(second argument of COMPLEX builtin functions or pseudo-variable must be REAL)

1A ILLEGAL COMPLEX ARGUMENT, REAL PART IS USED.
(the builtin functions ATAN (with two arguments), ATAND, COSD, ERF, ERFC, LOG10, LOG2, SIND, and TAND are defined only for real arguments)

1B ILLEGAL ARGUMENT TO BUILT-IN FUNCTION. SHOULD BE REAL, FIXED DECIMAL CONSTANT.
(warning: PL/I-F requires that certain arguments of the built-in functions BIT, CHAR, HIGH, LOW, and REPEAT be unsigned decimal constants. PL/C will take the argument as written)

1C RESULT SCALE FACTOR = number >127 IN MAGNITUDE. RESULT SCALED INCORRECTLY TO 127*SIGN(number)
(following the rules for PL/I expression evaluation, the scale factor ,q, of the result would be outside the permitted range -127 to 127. So that execution may be attempted, the result is scaled to the

```
          closest bound of the legal range. The value of the
          result will be incorrect)
1D      PROGRAM MAY LOOP IF THIS FORMAT IS EXECUTED
          (the FORMAT statement does not specify a data
          transmission format item. See explanation of error
          CG00)
1E      VARIABLE  iden  HAS A * BOUND OR LENGTH FIELD.
              10 IS USED.
          (only parameters in PL/C may have * bound or
          length)
1F      PARAMETER  iden  HAS A NON-* BOUND OR LENGTH FIELD.
          (parameters must have a * in this field in PL/C)
20      LOWER BOUND ON SUBSCRIPT  number
              OF  iden  EXCEEDS UPPER BOUND. (0:10) IS USED.
21      SPECIFIED P( number ) TOO LARGE. MAX
              PRECISION IS USED.
          (installation maximum precision, e.g. 31 for FIXED
          BINARY, 15 for FLOAT DECIMAL)
22      ARITHMETIC/BOOLEAN CONVERSION INVOKED
23      TOO MANY ERRORS DURING COMPILATION. EXECUTION
              SUPPRESSED
24      COMPILER ERROR DURING CODE GENERATION. PROGRAM ABORTED
          (a compiler error, see beginning of Section C)
```

EX Prefix Errors during the Execution Phase

```
NO   ON  MESSAGE
     CODE

00   0004 PROGRAM RETURNS FROM MAIN PROCEDURE
01   0004 PROGRAM IS STOPPED
          (normal termination-- a STOP or EXIT has
          been executed)
02   0070 END OF FILE REACHED
          (the ENDFILE condition is raised. System action
          terminates the program.)
03   0300 EXPONENT OVERFLOW. RESULT IS SET TO 1.
04   0300 EXPONENT OVERFLOW. RESULT IS LEFT UNCHANGED.
05   0310 FIXED POINT OVERFLOW
          (low order digit set to 1)
06   0310 FIXED DECIMAL OVERFLOW.
07   0310 NUMBER TOO LARGE TO CONVERT TO FIXED BINARY. 1 IS USED.
08   0320 FIXED-POINT QUOTIENT TOO LARGE. PROBABLE DIVISION BY 0.
              RESULT IS SET TO 1.
09   0320 FIXED-POINT QUOTIENT TOO LARGE. PROBABLE DIVISION BY 0.
              RESULT IS LEFT UNCHANGED.
0A   0320 FLOATING-POINT DIVISION BY 0. RESULT IS SET TO 1.
0B   0320 FLOATING-POINT DIVISION BY 0. RESULT IS LEFT UNCHANGED.
0C   0330 EXPONENT UNDERFLOW. RESULT IS SET TO 0.
0D   0330 EXPONENT UNDERFLOW. RESULT IS LEFT UNCHANGED.
0E   0340 SIZE RAISED. RESULT UNCHANGED.
          (occurs when the value of an expression is assigned
          to a variable whose precision is too small to hold
          the value. In PL/C, no left-truncation occurs.
          Instead the computed value is assigned to the
          variable, regardless of its declared precision.)
0F   0340 SIZE CONDITION RAISED DURING CONVERSION. RESULT IS
              SET TO 0.
10   0340 SIZE RAISED DURING STRING-TO-ARITHMETIC CONVERSION.
              VALUE USED IS number
11   0340 NUMBER TOO LARGE TO CONVERT TO SPECIFIED BIT STRING.
              (SIZE CONDITION)  NUMBER IS number
              STRING USED IS string
12   0340 RESULT OF BIT-TO-ARITHMETIC CONVERSION GREATER THAN
              2**56-1. (SIZE CONDITION)  STRING IS string
              VALUE USED IS number
13   0341 NUMBER TOO LARGE FOR FIELD. TRUNCATED ON LEFT.
              FULL FIELD WOULD BE string
          (in a PUT statement, the value is too large to fit
          in the specified field (for EDIT) or the field
          implied by the attributes of the item (for LIST).
          Signs and digits are lost on the left as in PL/I.
          The message indicates the full field before
          truncation.)
14   0350 INDEX OF SUBSTRING < 1 (number)
          (second argument of SUBSTR is less than one)
15   0350 INDEX OF SUBSTRING > STRING LENGTH (number)
          (second argument of SUBSTR is greater than
          the length of the first argument)
16   0350 LENGTH OF SUBSTRING < 0 (number)
          (value of third argument of SUBSTR is negative.
          It is replaced by 0.)
17   0350 SUBSTRING REQUESTED RUNS OVER END OF STRING
19   0520 SUBSCRIPT number OF iden IS OUT OF BOUNDS (number).
              number IS USED.
1A   0602 TOO MANY CHARACTERS FOLLOWING CLOSING QUOTE. ALL ARE
              IGNORED. FIELD IS string
1B   0603 TOO MANY DIGITS IN NUMBER, PRECISION LOST.
              STRING IS string
1C   0604 TOO MANY EXPONENT DIGITS, EXTRA DIGITS IGNORED.
              STRING IS string
1D   0605 INVALID CHARACTER(S) IN FIELD. 0 USED FOR EACH.
              ORIGINAL STRING IS string
              FIRST BAD CHARACTER IS string
          (the CONVERSION condition has been raised)
1E   0615 ILLEGAL CHARACTER(S) IN CHARACTER-TO-BIT CONVERSION.
              0'S USED.
1F   0900 ATTEMPT TO USE MATH BUILTIN FUNCTION IN "CALL"
              STATEMENT. STATEMENT IGNORED.
20   0901 iden REFERENCED RECURSIVELY. RECURSIVE ATTRIBUTE
              HAS NOW BEEN APPLIED.
          (indicated PROCEDURE is being used recursively
          but did not have RECURSIVE option)
21   0902 iden HAS IMPROPER LENGTH (number). 80 IS USED.
          (length is less than zero or greater than 256 and
          violates a PL/C restriction)
22   0903 LOWER BOUND ON SUBSCRIPT number OF iden
              EXCEEDS THE UPPER BOUND. (1:10) IS USED.
          (expressions for array bounds are evaluated before
          any statements in the block in which the array is
```

```
          declared are executed.  Variables used in these
          expressions must be initialized in an outer block)
23  0904 RETURN FROM iden VIA STMT line DOESN'T RETURN
             A VALUE AS EXPECTED IN STMT line . 0 IS USED.
24  0905 RETURN FROM iden VIA STMT line REQUIRES
             ILLEGAL CONVERSION. 0 IS USED.
          (PL/C restriction, PL/I-F would convert)
25  0906 RETURN FROM iden VIA STMT line RETURNS A VALUE
             TO "CALL" IN STMT line. VALUE IGNORED.
          (results would be unpredictable in PL/I-F)
26  0907 CALL TO iden FROM STMT line RETURNS VIA
             STMT line WITH STRING LONGER THAN DECLARED
             LENGTH. RETURNED LENGTH IS USED.
27  0908 CALL TO iden FROM STMT line RETURNS VIA
             STMT line WITH STRING SHORTER THAN DECLARED
             LENGTH. IT IS PADDED.
28  0909 BOUNDS OF iden DO NOT MATCH BOUNDS IN THE REST
             OF THE EXPRESSION
          (execution is terminated)
29  0910 iden HAS NOT BEEN ALLOCATED.
          (in a procedure invoked to initialize a variable a
          reference has been made to an array or structure or
          string which has not been allocated space. Variables
          are allocated and initialized in the order in which
          they are declared. See Section D.)
2A  0911 FORMAT LABEL IN GOTO
          (execution is terminated)
2B  0912 VALUE OF LABEL VARIABLE (iden IN STMT line) IS
             IN A CURRENTLY INACTIVE BLOCK.
          (execution is terminated)
2C  0913 iden INVOKED FOR INITIALIZATION IN STMT line
             TERMINATES VIA GOTO
          (execution is terminated)
2D  0914 iden IN STMT line IS IN A CURRENTLY
             INACTIVE ITERATIVE DO GROUP
          (execution is terminated)
2E  0915 SECOND ARGUMENT OF BIT/CHAR IS NOT POSITIVE. IMPLIED
             LENGTH IS USED.
2F  0916 STRING > 256 CHARACTERS LONG
          (PL/C limitation. Only the 256 left-most characters
          are retained)
30  0917 ATTEMPT TO ASSIGN INVALID BIT STRING TO DECIMAL FIXED
             DATA ITEM. 0 IS USED.
          (may occur in UNSPEC pseudo-variable, or during
          a READ statement. See Section G, pseudo-variables.)
31  0918 UNDEFINED ENTRY.
          (the procedure $UENTRY, supplied by PL/C
          to repair some semantic error, has been referenced)
32  0919 DELETED STATEMENT ENCOUNTERED
          (this message is produced during execution of a
          program when a statement deleted by an earlier
          phase of the compiler is encountered)
33  0920 UNDEFINED LABEL IN GOTO
          (execution is terminated)
34  0921 UPPER BOUND ON SUBSCRIPT FOR iden > 32767
             IN MAGNITUDE. 10 IS USED.
35  0922 LOWER BOUND ON SUBSCRIPT FOR iden > 32767
             IN MAGNITUDE. 1 IS USED.
36  0923 LABEL COUNTER OVERFLOW. IT IS RESET TO 0.
          (warning: a labeled statement has been executed
          more than 10 million times causing an internal PL/C
          counter to overflow.  This may indicate a loop in
          the program.)
37  0924 MULTIPLE INTERRUPTS, 2ND AND SUBSEQUENT ONES IGNORED
          (occurs on 360/91 and 370 only)
39  0925 RECORD I/O STRUCTURE VARIABLE iden CONTAINS VARYING
             STRINGS. MAXIMUM LENGTHS ARE USED.
          (only fixed length strings may be members of
          structures used by RECORD I/O)
39  0926 INVALID PARAMETER REFERENCE (OR COMPILER ERROR)
          (a parameter has been referenced which was not in
          the parameter list of the entry point used to call
          the procedure)
3A  0927 ATTEMPT TO USE AUTOMATIC ARITHMETIC-STRING CONVERSION.
          (an arithmetic variable or expression in an I/O
          list has been associated with a string format item
          or string data -- or vice-versa. The format item and
          corresponding list item are skipped.)
3B  0928 OUTPUT STRING TOO LONG. FIRST 32767 CHARACTERS USED.
3C  0929 INVALID BLANK FIELD IN GET EDIT. 0 USED.
3D  0930 DIMENSION SPECIFIED IN HBOUND, LBOUND OR DIM < 1
          (1 is used as the argument)
3E  0931 DIMENSION SPECIFIED IN HBOUND, LBOUND OR DIM > MAXIMUM.
             MAXIMUM IS USED.
3F  1002 ATTEMPT TO WRITE OVER END OF STRING. STATEMENT
             TERMINATED.
40  1002 ATTEMPT TO READ OVER END OF STRING. STATEMENT
             TERMINATED.
41  1018 CLOSING QUOTE MISSING IN INPUT FIELD: string
             QUOTE SUPPLIED.
42  3798 ONSOURCE/ONCHAR PSEUDO-VARIABLE USED OUT OF CONTEXT
          (ONCHAR or ONSOURCE may be changed by the
          program only when they have been set to point to a
          string in error, when the CONVERSION condition
          arises. At other times an attempt to change (assign
          to) either is an error)
43  3799 IMPROPER RETURN FROM CONVERSION ON-UNIT. SOURCE IS
             string
          (the CONVERSION ON- unit did not change the
          character which was in error)
44  0936 FEATURE NOT AVAILABLE IN THIS RELEASE
          (PL/I-F feature used that is not implemented in
          the current release of PL/C)
45  0937 iden IS AN ILLEGAL FORMAT LABEL.
          (the label referenced by the R( label ) format
          item is illegal. This may be:
             a) because it is not the label of a FORMAT
                statement, or
             b) because it labels a statement internal to some
                block other than the block containing the
```

```
            R( label )
        The remote format item is ignored)
46  0010 string IS AN ILLEGAL NAME
        (something other than an identifier was read during
        a GET DATA statement, where an identifier should
        have appeared. The NAME condition is raised.)
47  0938 INVALID FORMAT OPTION
        (an option in the format used with a GET or
        PUT EDIT statement appeared in an illegal context:
           A or B format: appeared without a field-width
               parameter on input
           COLUMN format: appeared without a target column
               parameter, or was used in GET/PUT
               STRING statement
           F format: appeared without a field-width
               parameter
           LINE format: appeared without a target line
               parameter
           LINE or PAGE: was used in a GET
               statement, in a PUT STRING, or a PUT
               FILE(X) where X was not a PRINT file
           P format: appeared- not implemented yet in PL/C
           SKIP format: was used in a GET/PUT STRING
               statement
           X format: appeared without a field-width
               parameter
        The format item and corresponding list item
        are dropped)
48  0939 INVALID FORMAT ITEM OPERAND
        (in formats E(W,Q), F(W,Q) or F(W,Q,P) either:
           a) a negative Q appeared on input; or
           b) on output, either 0>W, W>255, 0>Q, or
               Q>W
        The format item and corresponding list item
        are skipped)
49  0010 string IS NOT KNOWN TO PROGRAM
        (in a GET DATA statement, the name on the data
        card has not been used in the program. The NAME
        condition is raised. The data-card assignment is
        skipped)
4A  0010 INCOMPATIBLE STRUCTURE FOR iden
        (in a GET DATA statement a name in the input
        was qualified, although it was declared without sub-
        structures. Or, an unqualified name appeared in the
        data, although it was declared as a structure in the
        program. The NAME condition is raised. The
        data-card assignment is skipped)
4B  0010 iden IS NOT IN GET LIST
        (in a GET DATA statement, a name appeared in
        the input which was not in the data list. This error
        can arise for a qualified name if its first
        identifier (major structure identifier) is not in
        the data list. The NAME condition is raised.
        The data-card assignment is skipped)
4C  0010 ARRAY ERROR FOR iden
        (in a GET DATA statement, subscripts appeared
        on a name in the input, but the name was not
        declared as an array and may not be subscripted. The
        data-card assignment is ignored)
4D  0520 string BOUND ERROR. Number IS USED.
        (in a GET DATA statement, a subscript on a name
        in the input is out-of-bounds. The upper or lower
        bound is used, as indicated)
4E  0010 NO BOUNDS SPECIFIED FOR iden
        (in a GET DATA statement, no subscript appeared
        in the input following an array name. The
        data-card assignment is ignored)
4F  0081 CONFLICTING FILE ATTRIBUTES SPECIFIED OR IMPLIED.
            CODE= number
        (the codes are
           0: PUT to RECORD file (SYSPRINT will be used
                                  if possible
           1: GET from OUTPUT or RECORD file
           2: more than one of INPUT, OUTPUT, UPDATE
                                  specified
           3: STREAM file specifying non-CONSECUTIVE
                                  organization
           4: both RECORD and STREAM specified
           5: both DIRECT and SEQUENTIAL specified
           6: both DIRECT and CONSECUTIVE specified
           7: SEQUENTIAL CONSECUTIVE and KEYED specified
           8: DIRECT OUTPUT and INDEXED specified)
50  0084 FILE CANNOT BE OPENED. CODE= number
        (the operating system refuses to open a file.
        The codes are
           1: BLKSIZE not a multiple of LRECL (RECFM=F FB)
           3: missing DD card (or unknown reason)
           6: RECFM,LRECL,BLKSIZE not specified for
                                  non-PRINT file
           7: spanned records not supported
           8: BLKSIZE not large enough (RECFM=V or VB)
          13: auxiliary I/O not permitted at this
                                  installation)
51  0932 SYSTEM DATA SET CANNOT BE RE-ALLOCATED WHILE OPEN
            UNDER ANOTHER FILE.
        (an attempt has been made to open SYSIN or
        SYSPRINT while it is open under another filename)
52  0933 FILE NOT OPENED IN UNDEFINEDFILE ON-UNIT.
53  0934 INVALID ARGUMENT TO LINENO. Iden NOT A PRINT FILE.
54  0935 INVALID ARGUMENT TO COUNT. Iden NOT A STREAM FILE.
55  1004 string OPTION INVALID. FILE DOES NOT HAVE "PRINT"
            ATTRIBUTE.
        (the LINE and PAGE options are invalid in
        a PUT FILE(X), unless X has the PRINT
        attribute. The option is ignored)
56  1009 FILE CANNOT BE USED FOR STREAM INPUT.
        (file is open as a RECORD or OUTPUT file)
57  1009 FILE CANNOT BE USED FOR STREAM OUTPUT.
        (file is open as a RECORD or INPUT file.
        SYSPRINT will be used if possible.)
```

```
58  1009 FILE CANNOT BE USED FOR RECORD I/O.
            (file is open for STREAM I/O)
59  1009 FILE CANNOT BE USED FOR INPUT.
            (file is open for OUTPUT)
5A  1009 FILE CANNOT BE USED FOR OUTPUT.
            (file is open for INPUT)
5B  1009 I/O STATEMENT AND/OR OPTIONS INCOMPATIBLE WITH FILE.
               CODE= number
            (given for special RECORD I/O options. Codes are
               0: KEY/KEYTO/KEYFROM specified for non-KEYED file
               1: "KEY" not valid for this type of file
               2: "KEY" or "KEYFROM" required
               3: other incompatibility)
5C  0940 COMPILER ERROR - NO NAME FOR UNINITIALIZED VARIABLE AT
               OFFSET number
            (may occur if SUBSCRIPTRANGE is disabled and an
            uninitialized value is referenced. If this occurs
            when SUBSCRIPTRANGE is enabled it is a compiler
            problem)
5D  0941 iden HAS NOT BEEN INITIALIZED. IT IS SET TO string
5E  0942 FORMAT iden HAS INVALID CONDITION PREFIXES.
            (the conditions in effect for a FORMAT statement
            must be the same as those in effect for the EDIT
            statement which references the FORMAT statement.
            the conditions on the FORMAT statement are ignored
            and execution continues)

Note: For errors EX5F through EX6C
        see  IBM Document C28-6590 for exact formulae used.

5F  1509 rtn ABS(X) GE (2**50)*K; TAN(X) - K=PI AND IN TAND(X)
               -K=180. RESULT IS SET TO 1.
            (issued by TAN or TAND.
            TAN(X) is called directly by TANH(A+BI) and
            TAN(A+BI). The argument is too large in absolute
            value)
60  1513 rtn ABSOLUTE VALUE OF REAL ARGUMENT ( number ) IS
               > 175.366. RESULT IS SET TO 1.
            (issued by SINH or COSH)
61  1507 rtn ARGUMENT( number ) IS GREATER THAN PI*2**50 -
               .3537D+16. RESULT IS SET TO 1.
            (issued by COS or SIN.  COS(X) and/or SIN(X)
            are called by COSD, SIND, SIN(A+BI), COS(A+BI),
            SINH(A+BI), COSH(A+BI), and EXP(A+BI))
62  1501 rtn THE ARGUMENT IS NEGATIVE ( number ). RESULT IS
               SET SQRT(ABS(ARG)).
            (issued by SQRT(X). SQRT(A**2+B**2) is used to
            calculate ABS(A+BI), and various real SQRT calls
            are made in calculating SQRT(A+BI). In these
            indirect cases, message EX8D should not occur,
            but calculational errors might produce it)
63  1511 rtn BOTH ARGUMENTS ARE 0. RESULT IS SET TO 1.
            (issued by ATAN(Y,X) or ATAND(Y,X). ATAN(Y,X)
            is used in calculating LOG(A+BI), ATAN(A+BI),
            and (A+BI)**(C+DI))
64  1505 rtn ARGUMENT( number ) <= 0. RESULT IS SET TO 1.
            (issued by LOG(X). LOG(X) is called to compute
            LOG2(X), LOG10(X), (A+BI)**(C+DI), LOG(A+BI), and
            ATANH(Y). ATANH(Y) is in turn used in ATAN(A+BI)
            and ATANH(A+BI))
65  1559 rtn Z=+I OR -I IN ATAN(Z) OR  Z=+1 OR -1 IN ATANH(Z).
               RESULT IS SET TO 1+0I.
            (issued by ATAN(A+BI) or ATANH(A+BI))
66  1515 rtn ABSOLUTE VALUE OF ARGUMENT IS >=1. RESULT IS SET
               TO 1.
            (issued by ATANH(X). ATANH(X) is used in
            calculating ATANH(A+BI) and ATAN(A+BI))
67  1557 rtn Z1=0 AND IMAG(Z2) NE 0 OR REAL(Z2) LE 0. RESULT IS
               SET TO 1.
            (issued by (A+BI)**(C+DI))
68  1556 rtn IN COMPLEX EXPONENTIAL FUNCTION REAL ARGUMENT IS
               GREATER THAN 174.673. RESULT IS SET TO 1+0I.
            (issued by EXP(A+BI). EXP(A+BI) is used in
            calculating Z**W, when W or Z is complex)
69  1556 rtn IN COMPLEX EXPONENTIAL FUNCTION IMAGINARY ARGUMENT
               IS GREATER THAN PI*2**50 =.3534D+16. RESULT
               IS SET TO 1+0I.
            (issued by EXP(A+BI). See message EX93)
6A  1555 rtn Z=0 AND N LE 0 IN Z**N. RESULT IS SET TO 1+0I.
            (issued by X**Y)
6B  1505 rtn BOTH REAL AND IMAG ARGUMENTS ARE 0. RESULT IS SET
               TO 1+0I.
            (issued by LOG(A+BI). LOG(A+BI) is used in
            calculating Z**W, when W or Z is complex)
6C  1553 rtn ARGUMENT( number ) IS > 174.673. RESULT IS SET
               TO 1.
            (issued by EXP(X). EXP(X) is called in
            calculating ERF, ERFC, TANH, SIN(A+BI), COS(A+BI),
            SINH(A+BI), COSH(A+BI), and EXP(A+BI). EXP(A+BI)
            is in turn used in calculating Z**W, when W
            or Z is complex)
6D  0023 FILE name - ATTEMPT TO READ/WRITE RECORD OF
    0024        ZERO LENGTH.
6E  0021 FILE name - LENGTH OF VARIABLE( number ) ¬= LENGTH
    0022        OF RECORD( number ).
            (the number of bytes of storage occupied by the
            variable must equal the number of bytes in the
            record)
79          CONDITION (iden) SIGNALLED. NO ON-UNIT PENDING.
7A          iden SIGNALLED. "ERROR" RAISED AS STANDARD SYSTEM
               ACTION.
7B          KEY CONDITION RAISED. ONCODE= number.
7C          NORMAL RETURN FROM "ERROR" ON-UNIT. PROGRAM IS STOPPED.
7D          NORMAL RETURN FROM "FINISH" ON-UNIT. PROGRAM IS STOPPED
7E          ABOVE ERROR IS FATAL. PROGRAM IS STOPPED.
7F          NOT ENOUGH CORE. TRY LARGER REGION.
80          AUXILIARY I/O LIMIT EXCEEDED
               (see  AUXIO option in Section B)
81          ATTEMPT TO SWITCH FILE TO SYSPRINT HAS FAILED.
```

Section D. Summary of Differences between PL/C and PL/I

PL/C is a subset of PL/I. It is intended to be "upward compatible" with PL/I. A program that runs without error under the PL/C compiler should run under PL/I-F and produce the same results.

However, certain incompatible diagnostic features have been added to PL/C. If the programmer wishes to use these features for diagnostic runs under PL/C and still be able to run the same program under PL/I he must enclose such features in "pseudo-comments". If the programmer elects to use the incompatible PL/C MACRO feature it is impossible to preserve compatibility with PL/I.

PL/I-F Features not Included in PL/C

1. Direct-access auxiliary files.
2. Controlled and based storage, and list processing.
3. Multi-tasking.
4. Compile-time facilities, except for INCLUDE and an incompatible MACRO facility (see Sections I and J).
5. 48 character set option.
6. Message DISPLAY to the operator.
7. Sterling and picture data-types.
8. DEFINED and LIKE attributes.
9. A few built-in functions and pseudo-variables (see Section G).

Additional restrictions imposed by PL/C

1. 32 statement keywords and 6 auxiliary keywords are reserved and cannot be used as identifiers.
2. The names of built-in functions and pseudo-variables are not reserved and may be used as identifiers, but if they are to be used in this way they should be explicitly declared--contextual declaration of these particular identifiers may succeed (depending upon context) but will produce a warning message.
3. Parameters cannot be passed to the MAIN PROCEDURE of a PL/C program from the OS EXEC card.
4. String constants and comments must be contained in a single source card unless the PL/C NOBOUNDARY option is specified.
5. There is no implied conversion between string and coded-arithmetic data types in expressions. String constants cannot have repetition factors.
6. There are restrictions on the END, ENTRY, FORMAT, PROCEDURE, READ and WRITE statements.
7. There are restrictions on dimension, ENTRY, ENVIRONMENT, INITIAL, LABEL and length attributes.
8. Not all of the PL/I-F condition codes are used by PL/C and the default condition states under PL/C are not exactly the same as under PL/I-F.

Incompatible Features Added to PL/C

1. CHECK, NOCHECK, FLOW and NOFLOW statements; a FLOW condition; ONORIG, ONDEST, STMTNO built-in functions.
2. Diagnostic options on the PUT statement.
3. A built-in function to generate pseudo-random numbers.
4. Comments that are convertible to source text depending upon the first letter of their contents.
5. A text-replacement MACRO processor.

Differences In Internal Representation Of Data

Internally PL/C carries out all floating-point arithmetic operations in double-precision form, adopting user-specified precision only on output. This means that computation is often somewhat more precise than would be the case under PL/I-F. The result is usually a slight difference in the least-significant figures of results, but of course it is possible for the differences to become highly significant.

PL/C assigns a full word of storage to each FIXED BINARY variable and a double word of storage to each FIXED DECIMAL variable, regardless of the declared precision. This means that PL/C variables may hold values larger than their PL/I-F counterpart. However, the default state for the SIZE condition in PL/C is "enabled" so that situations in which PL/C would give different results from PL/I-F are detected.

Each bit in a PL/C bit-string is actually assigned an entire byte in storage. (Each PL/C string variable also has an eight byte control block called a dope vector so that an array of short strings takes a surprising amount of core.)

Decimal-base variables in PL/C are maintained internally in floating-binary form and converted on output.

This internal representation does not apply to record files, which are written in standard PL/I-F representation, and assumed to be in that representation when read. This means that PL/C and PL/I-F are compatible with respect to record files--files written by either compiler can be read by either.

Order Of Evaluation In DECLARE Statements

PL/I-F will reorder the evaluation of bounds and lengths and the initialization of variables so that, in the absence of circular dependencies, variables will be allocated and initialized before they are used to allocate or initialize other variables. PL/C uses a simpler strategy which depends upon the order in which DECLARE statements appear in the block, and the order in which variables are listed in a DECLARE statement:

1. First, all scalar arithmetic and label variables are given their initial value.
2. Then, proceeding in the order in which they are declared, strings, arrays and structures are allocated space and initialized. Any expressions in the bounds or length fields are evaluated before space is allocated.

After space has been allocated, the variable is initialized before processing the next variable in the order of declaration.

This strategy does not eliminate any allocation scheme available in PL/I-F but does require the programmer to order his declaration of variables to avoid the use of unallocated or uninitialized variables declared in the same block.

Dimensional Limits in the Compiler

The internal structure of the PL/C compiler is very different from that of the PL/I-F compiler and it was not feasible to limit certain critical dimensions of the source program in exactly the same way. This means that there are probably some unusually large and complex programs that would be accepted by PL/C but would exceed some dimensional limit in PL/I-F. (The opposite is certainly true.) The compilation limits in PL/C are the following:

1. Maximum nesting of IF statements is 12.
2. Maximum static (syntactic) nesting of PROCEDURE, BEGIN and DO statements is 11.
3. Maximum nesting of factors in DECLARE is 6.
4. Maximum number of label prefixes on a single statement is 87.
5. Maximum depth of parenthesis nesting in expressions is 14.
6. Maximum number of identifiers in a factor or structure in DECLARE is 88.
7. No single expression can contain more than 256 symbols.

These limits are fixed by the structure of the compiler and cannot be relieved by increasing the core made available to the compiler. In most other respects the compiler's limits are related to the amount of core available--for example, length of program and size of arrays. In these cases when the compiler indicates that a limit has been exceeded the user can resubmit the program with a larger region. The error messages in Section C indicate which errors involve fixed limits, and which can be alleviated with additional core.

Section E. Statements

The PL/I-F statements that are not included in PL/C are: ALLOCATE, DELAY, DELETE, DISPLAY, FREE, LOCATE, REWRITE, UNLOCK, WAIT

The statements that are included in PL/C are listed below. Except as noted, these statements are equivalent to PL/I-F.

Assignment

1. There is no implied conversion between string and coded-arithmetic data types.

BEGIN

1. The ORDER and REORDER options are accepted by PL/C but are not effective-- they do not alter the object code that is generated.

CALL

1. The TASK, EVENT and PRIORITY options are not included.
2. Scalars may not be given as arguments for array or structure parameters.

CHECK (not a PL/I statement)

1. When CHECK is encountered the printing that results from the raising of the CHECK condition (which may have been suppressed by a previous NOCHECK statement) is resumed. If there has been no preceeding NOCHECK statement, the CHECK statement is ineffective. Note that the normal action is to do the printing that results from the raising of the CHECK condition, so that the NOCHECK statement is provided to override this normal action. This is the opposite of the situation for the FLOW condition.
2. Alternate forms have one or two control parameters:

   CHECK(exp1) or CHECK(exp1, exp2)

   Exp1 specifies the maximum number of times that the printing resulting from raising of the CHECK condition in the current block will appear. After the specified number of instances NOCHECK is automatically applied. exp2 gives the maximum number of times the printing of the CHECK condition will be permitted in each block dynamically entered from the current block. That is,

   CHECK(N,M) is equivalent to
   CHECK(N) in the current block and
   CHECK(M,M) as the first statement in every block entered from the current block.

   CHECK(N) is equivalent to
   CHECK(N) in the current block and
   CHECK as the first statement in every block entered from the current block.

   CHECK is equivalent to
   CHECK in the current block and
   CHECK as the first statement in every block entered from the current block.

   Each time that a CHECK statement is executed the controlling counters are reset to the new limiting values.

CLOSE

DECLARE, DCL

1. The following words are reserved and cannot be used as identifiers:

ALLOCATE BEGIN CALL CLOSE DECLARE DCL DO END ENTRY EXIT FORMAT FREE GET GO GOTO IF ON OPEN PROCEDURE PROC PUT READ RETURN REVERT REWRITE SIGNAL STOP WRITE

TO BY WHILE THEN ELSE

NO CHECK NOCHECK FLOW NOFLOW

All of the other keywords are available for use as identifiers, but note the requirement under 2 below.

2. The names of built-in functions and pseudo-variables are not reserved and may be used as identifiers, but if they are to be used in this way they should be explicitly declared--contextual declaration of these particular identifiers may succeed (depending upon context) but will produce a warning message. For example, HIGH can be used as a variable name, but it should be listed in a DECLARE statement. Explicit declaration as a label or entry-name is also accepted.

3. PL/C does not include all of the attributes of PL/I-F. See Section F.

4. The maximum depth of factoring is 6.

5. The maximum number of identifiers in one factor or one structure is 88.

DO

END

1. If a label (entry-name) follows END this must have been the first (left-most) label (entry-name) on a preceding BEGIN, DO or PROCEDURE statement to be effective.

ENTRY

1. Scalars may not be used as arguments for array or structure parameters.

2. As in PL/I-F if the RETURNS option is not given the default attributes of the returned value depend upon the first letter of the entry name. For entry names beginning with any letter I through N, the default attributes are REAL FIXED BINARY (15,0). For entry names beginning with any other alphabetic character, the default attributes are REAL FLOAT DECIMAL (6).

3. Like PL/I, but unlike previous releases of PL/C, a * may not be given as a length specification in the RETURNS option. (Use CHAR(256) VAR or BIT(256) VAR to declare a procedure which returns a string of arbitrary length.)

4. An entry-name cannot be identical to an identifier that has been declared earlier in the procedure that contains the ENTRY statement. PL/C will reject such an entry-name as a "multiple declaration" even though this is a valid PL/I construction.

EXIT

1. Since PL/C does not include multi-tasking the EXIT statement is exactly equivalent to the STOP statement.

FLOW (not a PL/I statement)

1. When FLOW is encountered the printing resulting from the raising of the FLOW condition is resumed. If there has been no preceeding NOFLOW statement, the FLOW statement is ineffective. Note that the normal action is not to do the printing that results from the raising of the FLOW condition, so that the FLOW statement is provided to override this normal action. This is the opposite of the situation for the CHECK condition.

2. Alternate forms have one or two control parameters:

FLOW(exp1) or FLOW(exp1, exp2)

with exactly the same interpretation as for CHECK.

3. The FLOW condition is raised by any action that potentially alters the normal sequential flow-of-control. That is, by the CALL, DO, GOTO, RETURN and IF statements, by any exceptional condition (except FLOW) which causes an ON-unit to be entered, and by in-line procedure references.

FORMAT

1. P or 'picture' format is not included.

GET

GO TO, GOTO

IF

1. The maximum nesting depth for IF statements is 12.

NOCHECK (not a PL/I statement)

1. When NOCHECK is encountered the printing that would normally result from the raising of the CHECK condition is suppressed.

NOFLOW (not a PL/I statement)

1. When NOFLOW is encountered the printing that results from raising the FLOW condition is suppressed.

(NOSOURCE)

1. This pseudo-statement of PL/C6.5 is no longer included.

Null

ON

1. The default state for all PL/C conditions except CHECK and FLOW is enabled. (This differs from PL/I-F where the default state for SIZE, STRINGRANGE and SUBSCRIPTRANGE is "disabled".) If SUBSCRIPTRANGE is disabled (by a NOSUBSCRIPTRANGE prefix) the integrity of the compiler cannot be guaranteed and batch operation is threatened. The particular installation may elect to permanently enable the SUBSCRIPTRANGE condition in order to avoid this difficulty.

2. The AREA, KEY and PENDING conditions are not included. Not all of the PL/I ON codes are included. See Section H.

3. An incompatible FLOW condition has been added to PL/C. See Section H.

OPEN

1. The following options are not included:

DIRECT KEYED UPDATE BUFFERED UNBUFFERED
BACKWARDS EXCLUSIVE TRANSIENT

2. The TITLE option is not included in DOS PL/C.

3. All PL/C files are EXTERNAL.

PROCEDURE, PROC

1. The TASK option is not included.
2. The RECURSIVE option is included. All PROCEDURES under PL/C are recursive, but unless the RECURSIVE option is specified, recursive use of the procedure will result in a run-time diagnostic message.
3. The ORDER and REORDER options are accepted, but are ineffective--they have no effect on the object code generated.
4. As in PL/I-F if the RETURNS option is not given the default attributes of the returned value depend upon the first letter of the entry name. For entry names beginning with any letter I through N, the default attributes are REAL FIXED BINARY (15,0). For entry names beginning with any other alphabetic character, the default attributes are REAL FLOAT DECIMAL (6).
5. Like PL/I, but unlike previous releases of PL/C, a * may not be given as a length specification in the RETURNS option. (Use CHAR(256) VAR or BIT(256) VAR to declare a procedure which returns a string of arbitrary length.)
6. Scalars may not be used as arguments for array or structure parameters.

PUT

1. In addition to the PL/I options the following have been added to PL/C:
OFF -- suppresses printing of execution output on SYSPRINT.
ON -- resumes printing of execution output on SYSPRINT.
FLOW -- displays the recent FLOW history.
SNAP -- displays the recent calling history.
ALL -- displays the current values of all automatic, scalar variables in the blocks active at the time of encounter, and the current values of all static or external scalar variables.
ARRAY -- same as ALL but includes values of arrays.
DEPTH(exp) -- specifies the depth of block nesting for which the display is to be produced. Used only with SNAP, ALL or ARRAY options.
2. ON or OFF must be the only option on the statement.
3. FLOW, SNAP, ALL and ARRAY can appear in any combination with each other, and with the standard SKIP, PAGE or LINE options of PL/I. They cannot be combined with FILE, STRING, LIST, DATA or EDIT options.

READ

1. Only the FILE, INTO and IGNORE options are included.

3. PL/C-written fixed-length records are fully compatible with PL/I-F. But note that LABEL variables written by PL/C- compiled programs can be meaningful only when read back in by the same PL/C program in the same job step.

RETURN

1. There is no implied conversion between string and coded arithmetic data types.

REVERT

SIGNAL

(SOURCE)

1. This pseudo-statement of PL/C6.5 is no longer included.

STOP

WRITE

1. Only the FILE and FROM options are included, and both must be present.

3. PL/C-written fixed-length records are fully compatible with PL/I-F. But note that LABEL variables written by PL/C- compiled programs can be meaningful only when read back in by the same PL/C program in the same job step.

Section F. Attributes

The PL/I-F attributes that are not included in PL/C are listed below:

AREA	EVENT	PACKED
BACKWARDS	EXCLUSIVE, EXCL	PICTURE, PIC
BASED	GENERIC	POINTER, PTR
BUFFERED, BUF	IRREDUCIBLE	POSITION, POS
CONTROLLED, CTL	KEYED	REDUCIBLE
DEFINED, DEF	LIKE	TASK
DIRECT	OFFSET	

The attributes that are included in PL/C are listed below. Except as noted they are equivalent to the attributes of PL/I-F.

ALIGNED
AUTOMATIC, AUTO
BINARY, BIN
BIT
BUILTIN
CHARACTER, CHAR
COMPLEX, CPLX
DECIMAL, DEC
dimension
1. *'s must be used for the bounds in all dimension attributes associated with a parameter. This includes the bounds in the dimension attributes used in the ENTRY attribute for an entry parameter. *'s may not be used for the bounds under any other circumstances.

ENTRY
1. Structure parameters cannot be declared in the list after ENTRY. This means an entry-name cannot be passed as an argument if any of its parameters is a structure.
2. In the absence of explicit attributes PL/C supplies a scalar parameter with FLOAT, DECIMAL and REAL attributes, where under the same circumstances PL/I-F makes no assumption as to attributes.
3. The bounds in all dimension attributes and the length in all BIT or CHAR attributes in the parameter list that follows ENTRY must be *'s.

ENVIRONMENT, ENV
1. OS options included: CONSECUTIVE, F, V, U, CTLASA, CTL360
2. DOS options included: CONSECUTIVE, F, V, U, MEDIUM, CTIASA, LEAVE, NOTAPEMK, NOLABEL, VERIFY

EXTERNAL, EXT
FILE
1. Only EXTERNAL files are permitted.

FIXED
FLOAT
INITIAL, INIT
1. Iteration factors, but not string repetition factors are allowed in the INITIAL list. This means that the phrase (x) (1) '---' in PL/I-F would have to be given as (x) ('---') in PL/C. The phrase (x) '---' is not allowed.

INPUT
INTERNAL, INT
LABEL
1. The optional list of statement-label-constants is not allowed.

Length
1. A * must be used for the length in all CHAR and BIT attributes associated with a parameter. This includes lengths in the ENTRY attribute. *'s may not be used for lengths under any other circumstances.
2. The length in a CHAR or BIT attribute contained in a RETURNS attribute or option must be non-* (except when within ENTRY).
3. Length cannot be greater than 256.

OUTPUT
precision
1. The maximum precision and default precision is the same under PL/C as in PL/I-F. These values are:

max	default	base-mode
15	(5,0)	DECIMAL FIXED
31	(15,0)	BINARY FIXED
16	(6)	DECIMAL FLOAT
53	(21)	BINARY FLOAT

PRINT
REAL
RECORD
1. Only a limited form of record I/O is included. See READ and WRITE statements.

RETURNS
1. The length in a CHAR or BIT attribute may not be a * (except when within ENTRY).

SEQUENTIAL, SEQL
STATIC
STREAM
UNALIGNED, UNAL
VARYING, VAR

Default Attributes

Default attributes for base, mode and type depend upon the identifier name, as in PL/I-F. For identifiers beginning with any letter I through N, the default attributes are REAL FIXED BINARY(15,0). For identifiers beginning with any other alphabetic character, the default attributes are REAL FLOAT DECIMAL(6). If BINARY or DECIMAL and/or REAL or COMPLEX are specified, FLOAT is assumed unless FIXED has been specified. If FIXED or FLOAT and/or REAL or COMPLEX are specified, DECIMAL is assumed unless BINARY has been specified.

Factoring

1. The maximum depth of factoring is 6.
2. The maximum number of identifiers in a factor is 88.

Structures

1. The maximum number of identifiers in a structure is 88.

Section G. Built-in Functions and Pseudo-variables

The following PL/I-F built-in functions are not included in PL/C. A + after the name indicates that a pseudo-variable of the same name is also not included.

Based storage:
ADDR, EMPTY, NULL, NULLO
Multi-tasking:
COMPLETION+, PRIORITY+, STATUS+
Array Generic:
POLY
String Generic:
STRING pseudo-variable
Condition:
ONKEY
Miscellaneous:
ALLOCATION

The following built-in functions and pseudo-variables are included in PL/C. Except as noted otherwise, all built-in functions perform exactly as do their PL/I-F counterparts. In other words, the values of, say SIN(X) in PL/C and in PL/I-F are identical floating-point numbers when the functions are applied to the same floating-point value of X. The methods used in PL/C are those described in "PL/I Computational Subroutines: C28-6590".

The names of built-in functions and pseudo-variables are not reserved and may be used as identifiers, but if they are to be used in this way they should be explicitly declared--contextual declaration of these particular identifiers may succeed (depending upon context) but will produce a warning message. For example, HIGH can be used as a variable name, but it should be listed in a DECLARE statement. Explicit declaration as a label or entry-name is also accepted.

Pseudo-variables:
SUBSTR, REAL, IMAG, COMPLEX, ONCHAR, ONSOURCE, UNSPEC

Incompatibility: The right-hand side of the assignment
UNSPEC(decimal-fixed-variable) = bit-string;
must be a bit string that represents valid 360 packed decimal data. (This is because decimal based variables in PL/C are maintained internally in floating-binary form, and thus a conversion from packed decimal is required. This conversion requires valid packed decimal data.)

Arithmetic Generic:
ABS, ADD, DIVIDE, MAX, MIN, MOD, MULTIPLY, SIGN, FLOOR, CEIL, TRUNC, COMPLEX, CONJG, IMAG, REAL, ROUND

Incompatability: FLOOR, CEIL (FLOAT scale) under PL/I-F treat all numbers of magnitude <16**(-16) as zero; under PL/C only true zeros are treated as zero.

Mathematical Generic:
EXP, LOG, LOG10, LOG2, ATAND, ATAN, TAND, TAN, SIND, SIN, COSD, COS, TANH, ERF, ERFC, SQRT, COSH, SINH, ATANH
String Generic:
BOOL, HIGH, INDEX, LENGTH, LOW, REPEAT, STRING, SUBSTR, TRANSLATE, UNSPEC, VERIFY

Note on UNSPEC: Although the PL/I-F and PL/C internal representation of certain data-types differ (see Section D), the PL/C and PL/I-F UNSPEC functions produce identical results (except for the packed decimal error case noted under pseudo-variables.) The PL/C UNSPEC is somewhat more complicated (and time-consuming to execute) than its PL/I-F counterpart, since the PL/C UNSPEC must convert to and from the PL/I-F representation.

Array Generic:
ALL, ANY, DIM, HBOUND, LBOUND, PROD, SUM

Incompatibility: In PL/C the built-in functions DIM, LBOUND and HBOUND operate without actually evaluating an array expression, if such an expression is given as the first argument. Since PL/I-F evaluates the expression before exercising the function there could be "side effects" under PL/I-F that will not occur in PL/C.

Type Conversion:
BINARY, BIT, CHAR, DECIMAL, FIXED, FLOAT, PRECISION

These functions cannot be used to convert between string and coded-arithmetic forms of data. This type of converson can be accomplished thru GET and PUT STRING statements.

Condition:
DATAFIELD, ONCHAR, ONCODE, ONCOUNT, ONFILE, ONLOC, ONSOURCE

PL/C does not include all of the ON codes of PL/I-F. See Section H.

Miscellaneous:
COUNT, DATE, LINENO, TIME

PL/C Built-in Functions Not Included In PL/I-F:
RAND, ONORIG, ONDEST, STMTNO

A built-in function to produce a sequence of pseudo-random numbers has been included in PL/C. There is no comparable built-in function in PL/I-F. When this function is used a PL/C program will not run under PL/I-F. Therefore a program that includes the use of this built-in function will receive a warning message from PL/C.

Definition RAND finds the next number in a pseudo-random number sequence in which the argument was the last element, and returns the next number to the point of invocation.

Reference RAND(x)

Argument The argument "x" may be an element or array expression of coded arithmetic type. It must be REAL. If it is not also FLOAT it will be converted to FLOAT by the RAND function. The value must be in the range $0 < x < 1$.

Result The value returned will be FLOAT with the same base (BINARY, DECIMAL) as the argument.

Usage RAND is normally used in an assignment statement:

```
X = RAND(X);
```

Repeated execution of such a statement will cause X to take on the successive values in a sequence of pseudo-random numbers uniformly distributed over the interval (0.0, 1.0), so long as the initial, or "seed" value of X was properly chosen. The "seed" value should be between 0 and 1, and have nine significant figures, the right-most of which is odd. This will maximize the period of the sequence. The generation method used in the PL/C RAND function is based on the method of Coveyou and Macpherson in ACM Journal 14(1967), pages 100-119.

The ONORIG and ONDEST built-in functions have no arguments and can be used only within an ON FLOW unit. ONORIG returns the statement number of the statement that caused the FLOW condition to be raised, and ONDEST returns the statement number of the statement that is the target of that transfer.

The STMTNO built-in function has a single argument giving the label of a source statement, and it returns the statement number of that statement. It is also intended primarily for use in an ON FLOW unit, but may appear in any context.

Section H. Conditions

The PL/I-F conditions not included in PL/C are:
AREA, KEY, PENDING

All of the other conditions are included, and the FLOW condition has been added. The PL/C conditions are equivalent to their PL/I-F counterparts, as described in Section H of the IBM Reference Manual C28-8201 except as noted below. All of the conditions can be given in ON, REVERT and SIGNAL statements. (The NO--- conditions of course cannot be used in these statements.) The conditions and acceptable abbreviations are listed below, with the default state underlined:

CHECK NOCHECK

The system default display of value for the CHECK condition is not identical to that of PL/I-F. When an element of a 'checked' array is changed PL/C displays only that particular element (rather than the entire array.) Similarly, when a member of a 'checked' structure is changed only that member is displayed (rather than the entire structure.) The timing of the PL/C display is also not exactly the same as that of PL/I-F. The PL/C display occurs immediately after the check condition is raised, rather than waiting until the end of the statement. Unlike PL/I-F the raising of the CHECK condition may be suppressed and restored dynamically within the block by means of the PL/C CHECK and NOCHECK statements.

CONVERSION CONV NOCONVERSION NOCONV

ENDFILE

ENDPAGE

ERROR

The PL/C ERROR condition is not entirely compatible with PL/I-F. The standard system action in PL/I-F is to raise the FINISH condition and stop. In PL/C the standard system action is to apply the automatic PL/C error correction and then continue execution--the FINISH condition is not raised (unless that particular PL/C "correction" terminates execution.) Normal return (if there is a pending ON unit) is compatible--both PL/C and PL/I-F signal FINISH and terminate execution.

FINISH

FIXEDOVERFLOW FOFL NOFIXEDOVERFLOW NOFOFL

FLOW NOFLOW

The FLOW condition has been added to PL/C to permit the dynamic monitoring of the flow-of-control. When the FLOW condition is enabled by a FLOW prefix, the condition is raised by any statement that potentially will alter the normal sequential flow of execution. These are the CALL, DO, GOTO, IF and RETURN statements, any exceptional condition (except FLOW) which would cause an ON- unit to be entered, and in-line procedure references.

The standard system action depends on the use of the FLOW and NOFLOW statements. If the condition is raised within the scope of a NOFLOW statement, the origin and

destination of the transfer are saved in a first-in-first-out queue which will hold up to 18 entries. A PUT FLOW statement will cause the contents of this queue to be printed in the form:

oooo -> dddd or nnn*(ooo -> ddd)

where oooo is the origin statement number, dddd is the destination statement number, and nnn is a count of the number of times the transfer was made in succession. If the condition is raised within the scope of a FLOW statement the origin and destination of the transfer are printed on SYSPRINT in the same format as used by the PUT FLOW statement. This may, of course, be replaced with a user-supplied ON FLOW unit. The ONORIG and ONDEST builtin functions are useful in such an ON- unit-- see Section G. The NOFLOW and FLOW statements will still serve to dynamically suppress and resume the printing that may result from the raising of the FLOW condition. The user should disable the FLOW condition in any ON FLOW unit he supplies (by using a NOFLOW prefix) to avoid constructing an infinite loop.

If the FLOW or NOFLOW prefix is applied to a PROCEDURE or BEGIN statement its scope is the entire block. If applied to a DO statement its scope is the DO group. Applied to other statements, its scope is the single statement.

NAME

OVERFLOW OFL NOOVERFLOW NOOFL

RECORD

SIZE NOSIZE (note default different from PL/I-F)

When the SIZE condition is raised the arithmetic results in PL/C will differ from those produced by PL/I-F. In particular, PL/I-F may truncate results on the left to the user-specified precision, while PL/C always retains the implementation-defined maximum precision.

STRINGRANGE STRG NOSTRINGRANGE NOSTRG

(note default different from PL/I-F)

SUBSCRIPTRANGE SUBRG NOSUBSCRIPTRANGE NOSUBRG

(note default different from PL/I-F)

The NOSUBSCRIPTRANGE prefix will not be effective unless the FREE option is in effect. Since out-of-range subscripts can damage the compiler and interfere with batch operation many installations elect to inhibit (override) this option.

TRANSMIT

UNDEFINEDFILE UNDF

UNDERFLOW UFL NOUNDERFLOW NOUFL

User-defined condition

ZERODIVIDE ZDIV NOZERODIVIDE NOZDIV

As noted in the ERROR condition, PL/C will attempt to correct runtime errors and continue. If the particular condition is disabled (i.e., a NO-prefix is in effect) the correction will be suppressed. If a user-supplied ON-unit is pending, the correction will be applied, but no error message will be given.

The ONCODE values for each error condition are given in Section C. These are the same as the PL/I-F codes wherever possible. In addition, error cases detected by PL/C which have no PL/I-F codes are given codes in the 900-999 range.

Section I. The MACRO Feature

(The PL/C MACRO feature was designed and implemented by Professor David Gries and Mr. Tom London.)

Macro Definition

A macro definition must be given within a "macro packet". A packet begins with a *MACRO card (* in column 1) and ends with a *MEND card (* in column 1). One or more macro definitions comprise a packet. There is no limit on the length of a packet, or on the number of packets that may be used, although the core available will restrict the total amount of macro text in a program. Packets may be inserted anywhere in the source program, but each macro must be defined before it is referenced. A macro definition has the form

```
<macro name> = <macro body> %;
```

or

```
<macro name>(formal-param-1,formal-param-2,...,
             formal-param-n) = <macro body> %;
```

The macro body may contain any text except the sequence "%;". No requirement is made regarding balanced quotes or partial comments. Care should be taken about card boundaries, since macro text is interpreted as a series of lines, each with its own indentation. Two separate lines will never be put on the same line during expansion, although it may be necessary to split a line to fit within source margins.

The macro name may consist of up to 255 characters, starting with an alphabetic or national character. Like all symbols in PL/C, macro names cannot be split over card boundaries. (This is independent of the BNDRY option.) The name must be distinct from all other identifiers used in the program. PL/C keywords may be used as macro names, but such names will not be recognized as macros in any text which is syntactically scanned before the definitions are processed.

The formal parameters are recognized only within the macro being defined. They may contain up to 31 characters, again starting with an alphabetic or national character. Formal parameters are PL/C symbols, and therefore cannot be split over card boundaries. Formal parameters are local to a macro definition, and will over-ride within that definition any other use of the symbol (identifier, keyword or macro name). They are only recognized when surrounded by any of the PL/C delimiters (including blanks and source-text boundaries). Parameters appearing within comments and strings are also recognized, although, again, only when properly delimited. Thus, if E is a formal parameter in the text

```
E = 12;    /* SET E TO UPPER BOUND */
```

the E's in SET and UPPER will not be considered parameter instances.

Macro Expansion

Within PL/C source text, macro calls are expanded when encountered during syntactic analysis. This means that macro calls inside of strings and comments will not be recognized. Macro text may contain further macro references, either explicity or through parameter substitution, which are expanded when encountered during the scan of the original expansion. Direct or indirect self-reference is illegal, since this will always lead to an infinite loop of macro expansion. A macro call is of the form

```
<macro name>
```

or

```
<macro name>(actual-param-1, actual-param-2,...,
             actual-param-n)
```

The second form may be split over a card boundary, except that the macro name itself must not be split. The number of actual parameters must equal the number of formal parameters in the macro definition. During expansion, each actual parameter is substituted for the corresponding formal parameter in the macro body before the syntactic analysis scan of the macro text.

Actual parameters are treated as pure text during substitution. That is, embedded parameters are not recognized; macro references are not detected until the macro text containing the parameter is syntactically scanned. Blanks surrounding actual parameters in the call are ignored. As a convenience, there are two additional rules governing the text of actual parameters:

(1) If a parameter begins with a single quote, all text up to and including the next single quote will be considered part of the parameter. (Two consecutive quotes within such text do not terminate the parameter, and are not compressed into one.)

(2) If an actual parameter begins with a left parenthesis, all text up to the matching right parenthesis is considered part of the parameter. The outer parentheses, however, are discarded from the text of the parameter before substitution.

For example, given the macro packet

```
*MACRO
 ASGN(P,Q,R) = SUBSTR(P,Q) = R; %;
*MEND
```

and the macro call ASGN(X,(10,3),'ABC''XYZ') the expansion would be SUBSTR(X,10,3) = 'ABC''XYZ';

Macro Display

Unless NOSOURCE is specified, macro definitions are always printed. Macro calls are printed by default; NOMCALL may be used to suppress this printing. Macro expansions are normally not printed; the MTEXT option will cause them to appear. (If NOMTEXT and NOMCALL are both used, the default options -- NOMTEXT and MCALL -- are applied.)

Section J. The INCLUDE Feature (not in DOS PL/C)

The *INCLUDE feature is used to incorporate card images from an external file into the source text (during compilation) or into the SYSIN data stream (during execution). The *INCLUDE control card is defined as follows:

```
*INCLUDE member-list [, member-list]*;
```

where a 'member-list' is either a single member-name, or an optional DDNAME followed by a parenthesized list of member-names. Member-names in the list are separated by commas. If a DDNAME is not specified an installation dependent DDNAME is used. The supplied default DDNAME is SYSLIB.

Syntax Rules:

a. *INCLUDE must be given in columns 1-8 of the input card (or card image) regardless of the SORMGIN option. The SORMGIN option does apply to the included cards (except for any control cards that might be included.)
b. Each 'ddname' and 'member-name' pair identifies the card images to be incorporated into the PL/C job. These card images must form a member of a partitioned data set (PDS).
c. A 'ddname' specifies the characters appearing in the name field of some DD card in the JCL for the PL/C job step. This DD card should define a PDS. The 'member-name' associated with the 'ddname' specifies a member of the PDS to be incorporated into the PL/C job. Both 'ddname' and 'member-name' should have no more than 8 characters. Names will be truncated to 8 characters if they exceed that length, and no error message will be given.
d. Blanks may appear anywhere except in a 'ddname' or a 'member-name'.
e. A *INCLUDE may appear anyplace in a PL/C job deck.

General Rules:

a. Members are incorporated into the PL/C job in the order in which they appear on the *INCLUDE card.
b. Included cards may contain any text normally acceptable to PL/C from cards -- including *PL/C, *PROCESS, *DATA and *INCLUDE control cards.
c. *INCLUDE cards may be nested. That is, included cards may contain *INCLUDE cards.
d. A PDS to be included must have fixed format records of logical record length <= an installation defined maximum. (Default is 100.) The PDS may be blocked.

Errors:

If the card images cannot be included from the designated member an error message is printed. If more ddname/member-name pairs remain unprocessed on the *INCLUDE card in error, they are processed after the error message is printed.

This error message could be generated by any of the following conditions:

1) No space remains in the PL/C region for buffers or DCBs. Try a larger region.
2) The logical record length is too large.
3) The file specified by ddname could not otherwise be opened by the system.
4) The member-name could not be found in the PDS defined by the ddname.
5) A permanent I/O error occurred while reading from the specified member-name. (In this case a SYNADAF message is printed.)

Examples of *INCLUDE cards:

```
*INCLUDE MEMB1; includes MEMB1 from SYSLIB.

*INCLUDE (MEMB1,MEMB2); includes MEMB1 and MEMB2
                                from SYSLIB.

*INCLUDE MEMB1, MEMB2; same as previous example

*INCLUDE MYLIB(MEMB1,MEMB2);
                    includes MEMB1 and MEMB2 from MYLIB.

*INCLUDE MEMB1,MYLIB(MEMB2),YOURLIB(MEMB3,MEMB4);
                        includes MEMB1 from SYSLIB,
                        MEMB2 from MYLIB
                        and MEMB3 and MEMB4 from YOURLIB.
```

Section K. Efficient Programming in PL/C

PL/C was designed to emphasize speed of compilation rather than execution. If execution is substantial it may be worthwhile running under the IBM compiler once the program has been thoroughly checked out. However, if a program is to be run under PL/C and the execution time is significant, there are a number of options and devices that can be employed to improve execution speed. These all have the effect of suppressing or disabling some of the diagnostic provisions that are normally compiled into PL/C programs. While this will improve execution speed it obviously reduces the degree of protection and the amount of information provided. Execution speed will be increased by each of the following (if these are not already the defaults at your installation):

1. Disable the FLOW condition. This is done by giving a (NOFLOW): prefix for each external procedure. Even if the FLOW events are not being printed (as a result of execution of a FLOW statement) the tracing code is present and active if the FLOW condition is enabled. (Similarly the CHECK condition must be disabled-- by removing CHECK prefixes. Suppressing printing with the NOCHECK statement does not eliminate the checking code.)

2. Disable the SUBSCRIPTRANGE condition. This is done by giving a (NOSUBSCRIPTRANGE): prefix on each external procedure. This is only possible if the FREE option is in effect, and some installations override this option since subscript testing is vital to ensure the integrity of the compiler. If you can run in FREE mode the elimination of subscript testing will make a substantial improvement in the execution of a program with frequent references to subscripted variables.

3. Disable the SIZE condition. This is done by giving a (NOSIZE): prefix on each external procedure.

4. Specify the NOUDEF option. This will eliminate the code required to test for use of uninitialized variables. All variables (including strings) will be initialized to (hex) zero.

In order to limit the amount of printing from a PL/C program consider the use of the CMPRS, NODUMP, FLAGE, NOHDRPG, NOOPLIST and NOSOURCE options. To reduce the amount of printing during execution of the program consider using the PUT OFF; and PUT ON; statements in sections that are not of current interest.

Because of the differences in internal representation in PL/C there are certain operations that are relatively inefficient (in comparison to PL/I-F). These are record I/O, bit-string operations, and the UNSPEC and TRANSLATE built-in functions. To the extent that a program can avoid use of these features its execution speed relative to PL/I-F will be improved.

Section L. References and Textbooks

IBM PL/I Publications

IBM System 360 PL/I Reference Manual, C28-8201

IBM System 360 Operating System, PL/I-F Programmer's Guide, C28-6594

IBM System 360 PL/I Computational Subroutines, C28-6590

PL/C References

PL/C Release 7 Installation Instructions, Department of Computer Science, Cornell University

Conway, R. W. and T. R. Wilcox: "The Design and Implementation of a Diagnostic Compiler for PL/I", Communications of the ACM, March 1973.

Moore, C. G. and R. W. Conway: "PL/CT - An Interactive Terminal Version of PL/C", Department of Computer Science and Office of Computer Services, Cornell University

Morgan, H. L.: "Spelling Correction in Systems Programs", Communications of the ACM, Feb. 1973.

Wagner, R. A.: "Common Phrases and Minimum Space Text Storage", Communications of the ACM, March 1973.

Wilcox. T. R.: "Code Generation in PL/C", Research Report No. 70-89. Department of Computer Science, Cornell University

Wolfe, J. M.: "User's Guide to PL/C", Department of Computer and Information Science, Brooklyn College of the City University of New York. (To be published by Addison-Wesley.)

Textbooks Related Directly to PL/C

Bohl, Marilyn and Arline Walter: Introduction to PL/I Programming and PL/C, SRA, 1973.

Conway, Richard and David Gries: An Introduction to Programming, A Structured Approach Using PL/I and PL/C, Winthrop, 1973.

Davidson, Melvin: PL/I Programming with PL/C, Houghton Mifflin, 1973.

Hughes, Joan K.: PL/I Programming, Wiley, 1973.

Kennedy, Michael and Martin B. Solomon: Eight Statement PL/C plus PL/One, Prentice-Hall,1972.

Kochenburger, Ralph J. and Carolyn J. Turcio: Introduction to PL/1 and PL/C Programming, Hamilton, 1974.

Smith, Cecil L. and Paul W. Murrill: PL/I Programming with PL/C, Intext Educational Publishers, 1973.

Sprowls, R. Clay: PL/C: A Processor for PL/I, Canfield 1972.

Weinberg, G. M. et al: Structured Programming Using PL/C, Wiley, 1973.